Two-Max-Clique

Giacomo Arienti giacomo.arienti2@studio.unibo.it
Azael Garcia Rufer azael.garciarufer@studio.unibo.it

Introduction

This presentation outlines the process of adapting a MiniZinc model for the max clique problem to solve a more complex variant: finding two disjoint cliques that maximize the total number of selected nodes. We will discuss the original model, the modifications made, and the final results of our work.

Inspiration

Our journey began with an article titled 'Optimizing Clique Problems with Constraint Programming' sourced from IEEE. This article provided a detailed model for solving the max clique problem using MiniZinc, a high-level constraint modeling language. It outlined how to identify the largest clique within a graph, leveraging binary decision variables and adjacency constraints.

What for?

- **Social Network Optimization**: Identifying disjoint communities while maximizing the number of connections.
- **Communication Systems Design**: Enhancing network resilience by analyzing autonomous subgroups.
- **Genomics and Bioinformatics**: Detecting non-overlapping functional subsets in gene interaction graphs.
- **Logistics and Supply Chain**: Optimally allocating resources in disjoint scenarios to reduce overlaps.

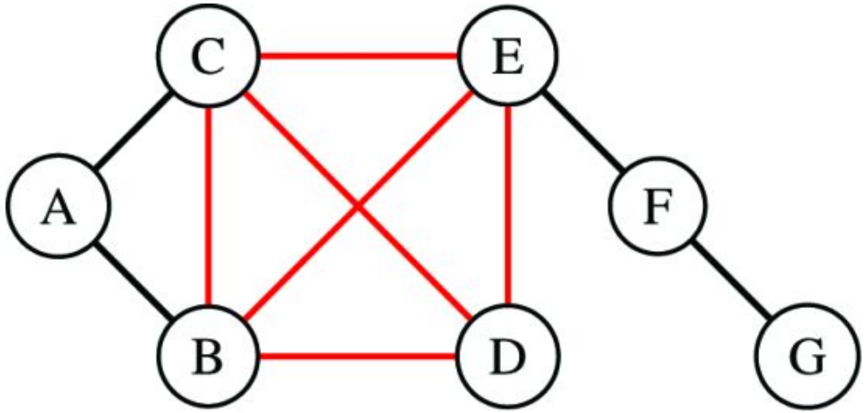
Understanding the original Model

Graph Representation:

- The input graph is represented by an **adjacency matrix**, g[i, j], where:

    - g[i, j] = 1 means there's an edge between nodes i and j.

    - g[i, j] = 0 means no edge exists.



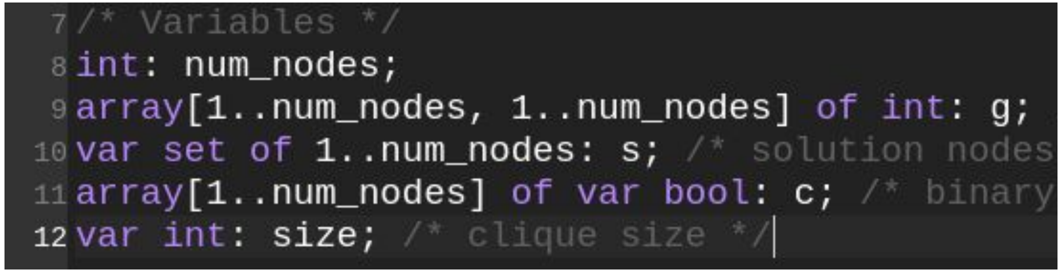
Understanding the original Model

Variables:

- Binary decision variables: if the node should be part of the clique.

- Set variable: represents the set of nodes included in the clique.

- Size variable: tracks the number of nodes in the clique, calculated as the sum of c[i].

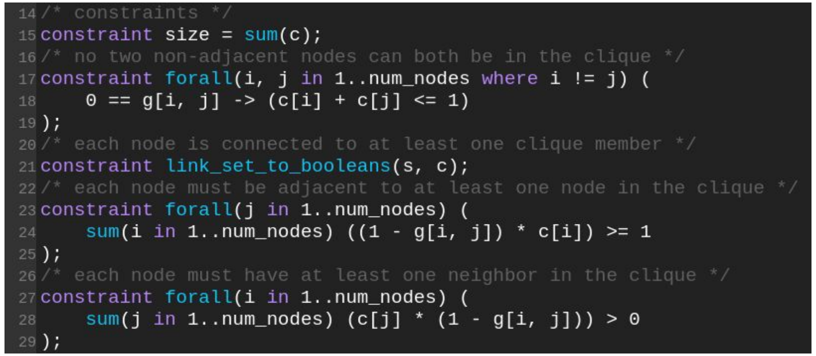


```
/* Variables */
int: num_nodes;
array[1..num_nodes, 1..num_nodes] of int: g;
var set of 1..num_nodes: s; /* solution nodes
array[1..num_nodes] of var bool: c; /* binary
var int: size; /* clique size */
```

Understanding the original Model

Constraints:

- Clique validity:
    - If two nodes are not adjacent (g[i, j] = 0), they cannot both be in the clique.
- Set-Binary mapping:
    - Each node in the graph must have at least one neighbor in the clique

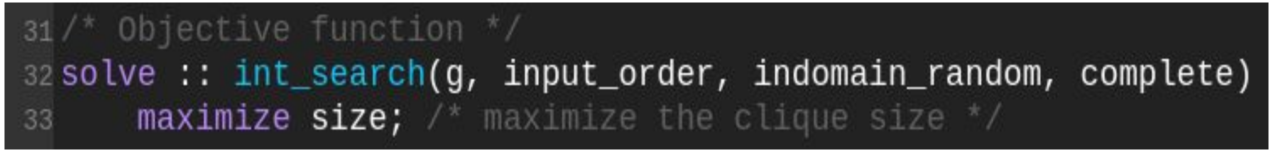


```
/* constraints */
constraint size = sum(c);
/* no two non-adjacent nodes can both be in the clique */
constraint forall(i, j in 1..num_nodes where i != j) (
    0 == g[i, j] -> (c[i] + c[j] <= 1)
);
/* each node is connected to at least one clique member */
constraint link_set_to_booleans(s, c);
/* each node must be adjacent to at least one node in the clique */
constraint forall(j in 1..num_nodes) (
    sum(i in 1..num_nodes) ((1 - g[i, j]) * c[i]) >= 1
);
/* each node must have at least one neighbor in the clique */
constraint forall(i in 1..num_nodes) (
    sum(j in 1..num_nodes) (c[j] * (1 - g[i, j])) > 0
);
```

Understanding the original Model

Objective function:

- The goal of the model is to **maximize the size of the clique**, which is represented as:



```
/* Objective function */
solve :: int_search(g, input_order, indomain_random, complete)
    maximize size; /* maximize the clique size */
```

- This tells the solver to find a solution where the sum of selected nodes (c[i]) is the largest possible.

Preparing the data for Adaptation

Before diving into the model modifications, we needed a reliable way to prepare input data. This involved creating a script to process adjacency matrices:

1. Convert graph representations, such as edge lists, into **adjacency matrices**.
2. Validate the data for symmetry and ensure it aligned with MiniZinc's input format.

Adapting the model for two Disjoint Cliques

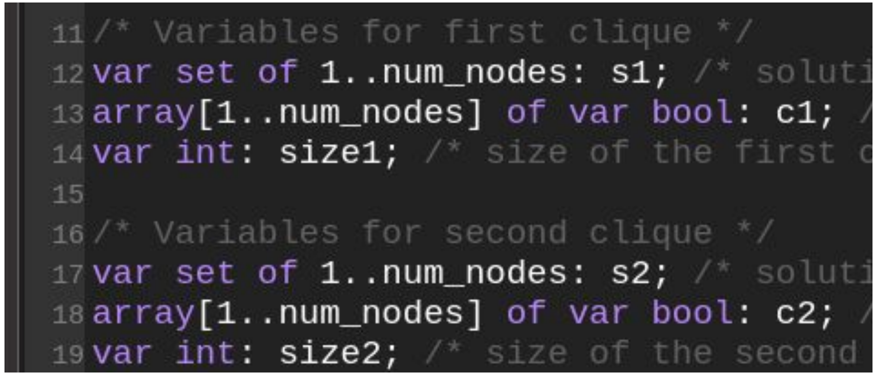
The core challenge was adapting the original max clique model to solve for two disjoint cliques. We approached this by breaking the problem into three key steps.

Adapting the model for two Disjoint Cliques

1. Duplicating variables:

- We introduced separate decision variables (c1 and c2) and sets (s1 and s2) for the two cliques.

- Corresponding size variables (size1 and size2) tracked the size of each clique.



```
/* Variables for first clique */
var set of 1..num_nodes: s1; /* soluti
array[1..num_nodes] of var bool: c1; /
var int: size1; /* size of the first c

/* Variables for second clique */
var set of 1..num_nodes: s2; /* soluti
array[1..num_nodes] of var bool: c2; /
var int: size2; /* size of the second
```

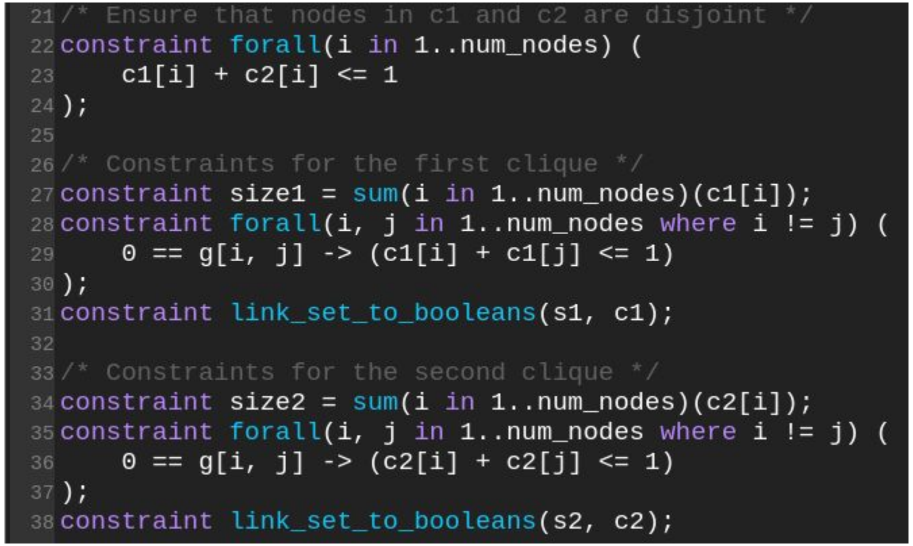
Adapting the model for two Disjoint Cliques

2. Modifying constraints:

- Constraints from the original model were applied separately to c1 and c2 to ensure each clique was valid.

- A new constraint, c1[i] + c2[i] <= 1, was added to enforce disjointness, ensuring no node could belong to both cliques.

Adapting the model for two Disjoint Cliques

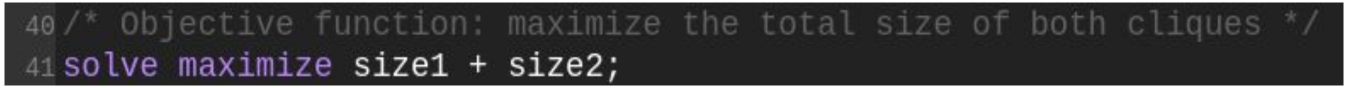


```
/* Ensure that nodes in c1 and c2 are disjoint */
constraint forall(i in 1..num_nodes) (
    c1[i] + c2[i] <= 1
);

/* Constraints for the first clique */
constraint size1 = sum(i in 1..num_nodes)(c1[i]);
constraint forall(i, j in 1..num_nodes where i != j) (
    0 == g[i, j] -> (c1[i] + c1[j] <= 1)
);
constraint link_set_to_booleans(s1, c1);

/* Constraints for the second clique */
constraint size2 = sum(i in 1..num_nodes)(c2[i]);
constraint forall(i, j in 1..num_nodes where i != j) (
    0 == g[i, j] -> (c2[i] + c2[j] <= 1)
);
constraint link_set_to_booleans(s2, c2);
```

Adapting the model for two Disjoint Cliques

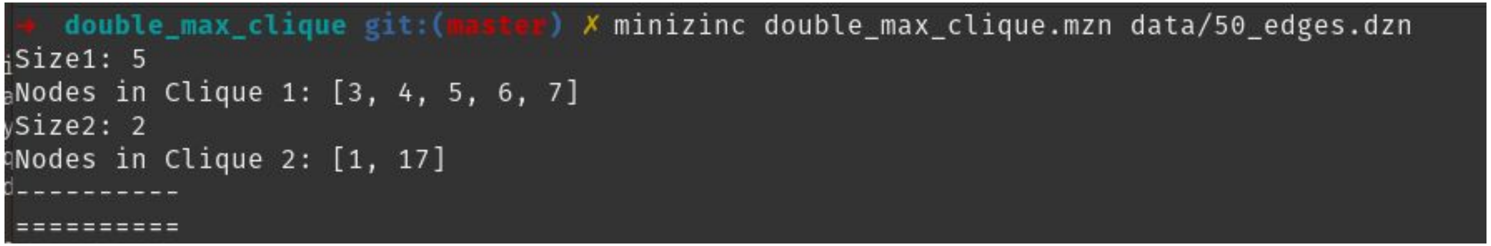
3. Updating the Objective Function:

- Instead of maximizing a single size, we aimed to maximize size1 + size2, reflecting the total number of nodes in the two cliques.



```
/* Objective function: maximize the total size of both cliques */
solve maximize size1 + size2;
```

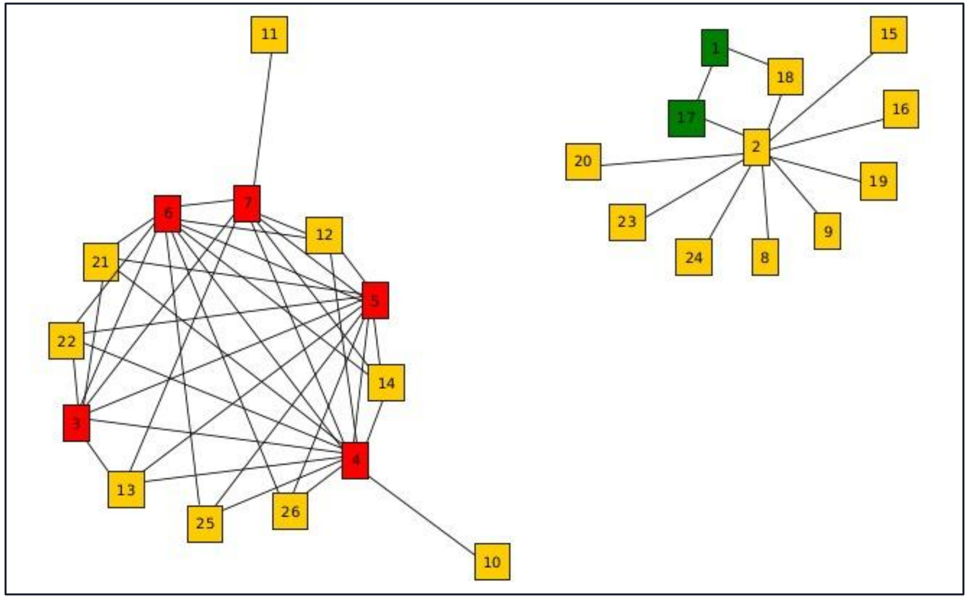
Final result

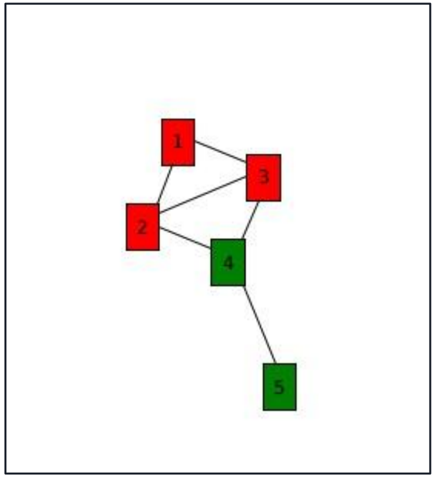


```
➜  double_max_clique git:(master) ✗ minizinc double_max_clique.mzn data/50_edges.dzn
Size1: 5
Nodes in Clique 1: [3, 4, 5, 6, 7]
Size2: 2
Nodes in Clique 2: [1, 17]
----------
==========
```

Graph visualization

To validate the obtained results we thought that the easiest way was to visualize it, so we created a script that given the input edge list will return a xlsx file that could be imported in **yED** and would let has create visual graphs.





What’s next?

- **Weighted Max Clique**: Analyzing graphs with weighted edges for more complex insights (e.g., connection importance).
- **Multigraph and K-Clique**: Studying disjoint cliques in multilayer graphs or graphs with diverse relationships.
- **Real Applications**: Implementing dynamic graph analysis for real use cases.

Thanks for your attention

https://github.com/giacomoarienti/minizinc-two-max-clique